Facultad de Ciencias

Diseño e implementación de los módulos de notificaciones y de cálculo de garantías de una aplicación de ventas por slot para una comercializadora de gas.

(Design and implementation of the notification and calculation of guarantees modules for a slot sales application for a marketer.)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autora: Sonia Pérez Sáiz

Directora: Marta Elena Zorrilla Pantaleón

Co-Director: Aurelio Montes Cobo

Septiembre - 2020

Tabla de contenido

Tabla de figuras

Agradecimientos

Antes de comenzar con este trabajo fin de grado, me gustaría aprovechar para agradecer a todas aquellas personas que me han ayudado a llegar hasta aquí.

En primer lugar agradecer a mi familia y amigos por apoyarme y animarme siempre a estudiar lo que me gusta y me llena aunque no siempre haya sido una elección fácil.

En segundo lugar me gustaría agradecer a todos y cada uno de los profesores que he tenido a lo largo de estos años de formación, por haberme enseñado tanto y haberme ayudado a crecer tanto profesional como personalmente. En especial a Marta Zorrilla Pantaleón por ayudarme a cerrar esta etapa con este trabajo.

Por último pero no menos importante, agradecer a la empresa en la que he cursado las prácticas, por la oportunidad, amabilidad y atención que han tenido conmigo durante estos meses.

Resumen

Este trabajo de fin de grado describe la realización e implementación de dos nuevas funcionalidades de una aplicación web ya existente en una comercializadora de gas natural. Dicha aplicación sirve para acceder y controlar diferentes datos como balances o medidas del gas de una forma sencilla y visual, gestionar contratos minoristas de venta de gas, el consumo y facturación, etc.

El primer módulo que se va a tratar en esta memoria es el de notificaciones. Este módulo se basa en informar al cliente de los problemas encontrados a través de correos electrónicos, como por ejemplo, la finalización de contratos, la acumulación de garantías a deber o que su solicitud de compra ha sido rechazada.

Por otro lado, el módulo de cálculo de garantías tiene por objeto establecer las condiciones que deben cumplirse en los contratos de las comercializadoras de gas para garantizar que el gas que se descarga y lleva por viaductos se consuma (se tenga prevista su venta) en los próximos dos meses. El cálculo se ha creado respecto a las características propias de la venta de slots.

Para ambas funcionalidades se desarrollaron los servicios, en la parte de backend, con Java y el framework de Spring, junto con las pantallas necesarias en la parte del frontend que se implementará en Javascript usando Angular.

Abstract

This end-of-degree work describes the realization and implementation of two new functionalities of an existing web application in a gas company. This application is used to access and control different data such as balances or gas measurements in a simple and visual way.

The first module to be dealt with in this report is that of notifications. This module is based on informing the client of the problems encountered through emails, for example, the termination of contracts, the accumulation of guarantees due or that their purchase request has been rejected.

On the other hand, the guarantee calculation module aims to establish the conditions that must be met in the contracts of the gas marketers to guarantee that the gas that is discharged and carried through viaducts is consumed (its sale is planned) in the next two months. The calculation has been created with respect to the characteristics of the sale of slots.

For both functionalities the services were developed, in the backend part, with Java and the Spring framework, along with the necessary screens in the frontend part that were developed in Javascript using Angular.

1. Introducción

En este primer apartado se va a describir los objetivos, organización y metodología seguida para el desarrollo de la presente memoria.

1.1. Antecedentes

Este TFG ha sido desarrollado en un proyecto de una empresa privada. Dicho proyecto esta solicitado por una entidad gasista, que se encarga, entre otras cosas, de la construcción de infraestructuras como gasoductos o almacenes subterráneos además de tener una buena gestión técnica del sistema de gas. Para conseguir esa gestión, se hace uso de una aplicación propia de la empresa, específica para dicho propósito. La aplicación solo es accesible desde la intranet del cliente y es necesario un usuario y una contraseña para iniciar sesión.

La aplicación se basa en estándares y conceptos como J2EE y SOA (Arquitectura Orientada a Servicios), lo cual facilita su uso y su entendimiento a la hora de desarrollar los sistemas. También es más eficiente el paso final a producción.

Estos estándares, metodologías y conceptos tienen como finalidad garantizar que el desarrollo de aplicaciones se realice de acuerdo con las especificaciones del diseño técnico y de acuerdo con los criterios de calidad del cliente.

La tarea de la empresa donde yo he estado era el desarrollo y mantenimiento de dicha aplicación, para cumplimentar con todas las necesidades que la comercializadora de gas necesite.

1.2. Objetivos

La aplicación proporciona funciones para el control del estado de las infraestructuras de la empresa, control y registro de las empresas comercializadoras que usan sus infraestructuras así como diferentes tipos de estimaciones.

En este proyecto se describen dos nuevas funcionalidades que se van a implementar en dicha aplicación.

El primer módulo del que se va a hablar es el de notificaciones. Este módulo se basa en informar al cliente a través de correos electrónicos de los problemas encontrados en cualquier ámbito, contratos, facturas, pagos,… Algunos ejemplos de esos problemas serían: la finalización de contratos, la acumulación de garantías a deber o que su solicitud de compra ha sido rechazada.

El segundo módulo que se va a tratar es el del cálculo de garantías que tiene por objeto establecer las condiciones que deben cumplirse en los contratos de las comercializadoras de gas para garantizar que el gas que se descarga y lleva por viaductos se consuma (se tenga prevista su venta) en los próximos dos meses. Debido a la nueva funcionalidad de venta de slot hay que adaptar y reprogramar el mismo.

1.3. Organización y metodología

Dentro de este proyecto la empresa cuenta con varios equipos de trabajo repartidos en distintas ciudades, Madrid, Santander y Sevilla. Cada uno de ellos con unas tareas diferentes dentro de la misma aplicación.

Madrid se centra en el trato más cercano con el cliente. Realizan una función de enlace entre los equipos de trabajadores y el cliente, ocupándose de las mejoras o cambios que puedan surgir en el desarrollo así como de ir informando al cliente del avance realizado cada semana. Este equipo por tanto, al ser el nexo de unión es el que cuanta con el *Team Leader*.

El equipo de Sevilla realiza un trabajo similar al de Santander, ambos se centran en el desarrollo del código. No voy a entrar en detalle de las tareas realizadas por el equipo de Sevilla pues no es al que yo pertenezco.

Por último el equipo de Santander está formado por un analista y 2 programadores, a los cuales me he unido durante estos meses para la realización de las nuevas funcionalidades.

1.4. Estructuración de la memoria

A lo largo de este documento se describen todas las fases propias de la ingeniería de software seguidas para el diseño, implementación y despliegue de los módulos a realizar.

En el apartado 2 se enumeran y describen todas las tecnologías utilizadas. En el apartado 3, se presentan los requisitos que deben satisfacerse para la finalización de los módulos. En el apartado 4, se describe el diseño a seguir. En el apartado 5, se detalla la implementación de la solución con las capas internas. En el apartado 6, se exponen las distintas pruebas llevadas a cabo para la comprobación del correcto funcionamiento, tanto de los nuevos módulos como de toda la aplicación tras las nuevas incorporaciones.

Y para terminar en el apartado 7, fuera de las fases propias de la ingeniería de software se añade un apartado de conclusiones y futuros trabajos, en donde se resume el trabajo realizado y se sugieren funcionalidades que se podrían implementar como continuación a este proyecto para próximos trabajos.

2. Tecnologías y herramientas utilizadas

En este apartado se describen las tecnologías y herramientas utilizadas en el desarrollo de los nuevos módulos.

Java

Java [1] es uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones web. Este lenguaje de programación orientado a objetos fue desarrollado originalmente por James Gosling, de Sun Microsystems aunque actualmente pertenece a Oracle. Las aplicaciones Java son compiladas a bytecode, que puede ejecutarse en cualquier máquina virtual de Java (JVM). Dentro de sus diferentes versiones, en este proyecto se utiliza la versión 1.6 debido a que la aplicación tiene varios años de desarrollo.

Spring

Spring [2] es un framework para el desarrollo de aplicaciones java que cuenta con código abierto. Proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas, en cualquier tipo de plataforma de implementación. El *framework* fue lanzado inicialmente bajo la licencia Apache 2.0. El corazón de Spring Framework es su contenedor de inversión de control. Su trabajo es instanciar, inicializar y conectar objetos de la aplicación, además de proveer una serie de características adicionales disponibles en Spring a través del tiempo de vida de los objetos.

Maven

Apache Maven [3] es una herramienta de software para la gestión y comprensión de proyectos java. Basado en el concepto de modelo de objeto de proyecto (POM) para describir el proyecto de software a construir, sus dependencias con otros módulos y componentes externos, y el orden de construcción de los elementos. Maven puede administrar la construcción, los informes y la documentación de un proyecto a partir de una información central. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Soap UI

SoapUI [4] es una herramienta, desarrollada en java, para la realización de pruebas a aplicaciones con arquitectura orientada a servicios (SOA) y transferencia de estado representacional (REST). Su trabajo se organiza en proyectos, un proyecto puede contener cualquier cantidad de pruebas funcionales, pruebas de carga y simulaciones de servicio requeridas para sus propósitos de prueba. Soporta múltiples protocolos y posee una versión de código abierto y otra versión de pago.

Svn

Tortoise SVN [5] es un sistema de control de versiones usado para que varios desarrolladores puedan trabajar en un mismo proyecto de manera remota y ordenada. Tiene una arquitectura cliente servidor con controles de concurrencia para cuando varios desarrolladores están trabajando en el mismo archivo. En el servidor se monta un repositorio SVN donde se registran los cambios y los logs que se van generando. El cliente se baja un copia de una versión (normalmente la última) a partir de la cual puede realizar sus cambios y volver a subirlo al servidor para que otro desarrollador pueda acceder a ellos.

MobaXterm

MobaXterm [6] es una terminal para Windows que permite acceder de manera eficiente a servidores remotos a través de diferentes redes o sistemas. Esta interfaz de usuario intuitiva proporciona todas las herramientas de red remotas importantes y comandos Unix en un único archivo exe portátil.

Oracle WebLogic

WebLogic [7] es un servidor de aplicaciones Java, nativo en la nube que centraliza servicios de aplicaciones, como la funcionalidad del servidor web, los componentes empresariales y el acceso a los sistemas empresariales backend. También puede automatizar muchas tareas a nivel de sistema ahorrando así tiempo de programación.

Oracle Sql developer

SQL Developer [8] es un entorno de desarrollo integrado y gratuito que simplifica el desarrollo y la gestión de Oracle Database en la nube y en implementaciones tradicionales. Además ofrece un completo desarrollo de sus aplicaciones, una hoja de trabajo para ejecutar consultas y scripts, una consola DBA

para gestionar la base de datos, una interfaz de informes, una solución completa del modelado de datos y una plataforma de migración con el fin de poder traer las bases de datos a Oracle.

Jira

Jira [9] es un Software diseñado para que todos los miembros de tu equipo de software puedan planificar, supervisar y publicar software de gran calidad. Crea historias de usuario, planifica sprints y distribuye tareas dentro del equipo. Conecta el trabajo de tu equipo con la hoja de ruta de tu producto. Cada equipo cuenta con un proceso único para lanzar software. Se puede utilizar un workflow predefinido o crear uno adaptado a la forma de trabajar de cada equipo.

Microsoft teams

Microsoft teams [10] es una plataforma unificada de Windows lanzada en 2017 para mejorar el trabajo en equipo en las empresas. Se basa en la comunicación y colaboración que permite realizar tareas desde cualquier lugar, como chatear, realizar reuniones a través de videoconferencias, llamar y trabajar en colaboración, entre otras. Además cuenta con la ventaja de poder descargarlo en diferentes dispositivos no solo en un ordenador, por ejemplo, en el teléfono móvil.

3. Especificación de requisitos

En este apartado de la memoria se detalla cómo se llevó a cabo la captura de requisitos y una descripción de las diferentes características que ofrecerá la aplicación al usuario tras el desarrollo del proyecto, así como la especificación de los requisitos funcionales y no funcionales, todo a través de un DDR (Documento de Definición de requisitos).

3.1. Captura de requisitos

La captura de requisitos fue realizada por el *team leader* y el analista de nuestro equipo. Esto se desarrolló a través de varias reuniones con el cliente donde hablaron sobre la incorporación de las nuevas funcionalidades y su validación. En caso de alguna duda posterior o cambio durante la implementación, como he dicho anteriormente, sería el equipo de Madrid el que se comunicaría con el cliente.

3.2. Requisitos funcionales

Los requisitos funcionales recogen las funcionalidades que el sistema debe proporcionar a los usuarios, su respuesta frente a situaciones específicas y, a veces, incluso indican cómo no debe comportarse éste.

En las siguientes tablas se detallan los diferentes requisitos del sistema que deben cumplir los módulos a desarrollar descritos anteriormente:

- Para el módulo de notificaciones los requisitos funcionales son:

Código	RF01
Modulo	Notificaciones
Nombre	Servicio agregado
Descripción	Para el caso de **servicio agregado**, se incluirán todos los contratos formalizados en una única notificación con las siguientes características: • Destinatarios: GTS, operador, comercializador • Asunto: Contratos formalizados correspondiente a la solicitud <<id_solicitus>> de <<actor_inicial>> • Cuerpo: Se han formalizado los contratos

Código	RF02
Modulo	Notificaciones
Nombre	Contrato implícito
Descripción	Para el caso de **contrato implícito**, se incluirán todas las referencias de los contratos en una única notificación. Tendrá el mismo contenido que la notificación anterior de servicio agregado.

Código	RF03
Modulo	Notificaciones
Nombre	Apertura de subasta

Descripción	Notificación para la apertura de subasta (solo para slots).
	• Destinatarios: Comercializadores con solicitud en la 2º ventana, GTS
	• Asunto: Apertura de subasta
	• Cuerpo: Tras finalizar la segunda ventana de modificación voluntaria de solicitudes, va a tener lugar la apertura de la subasta correspondiente a:
	  • Servicio: <<CARGA/DESCARGA>>
	  • Planta: <<planta de regasificación>>
	  • Fecha: <<fecha>>
	  • Fecha subasta: <<fecha subasta>>
	  • Hora inicio: <<hora inicio>>

Código	RF04
Modulo	Notificaciones
Nombre	Modificación de contrato por flexibilidad de slot
Descripción	Caso de modificación de contrato por flexibilidad de slot (no cambio de planta). • Destinatarios: GTS, operador, comercializador • Asunto: Modificación de contrato/s por flexibilidad de slot correspondiente a la solicitud <<id_solicitud>> de tipo <<tipo solicitud>>, de <<actor_inicial>> • Cuerpo: Informarle que tras la aceptación de la solicitud <<id_solicitud>> de tipo <<tipo solicitud>>, de <<actor_inicial>> se han actualizado los contratos

Código	RF05
Modulo	Notificaciones
Nombre	Cambio de planta
Descripción	Cambio de planta suministradora de gas para antiguos y nuevos contratos • Destinatarios: GTS, operador origen, comercializador

	• Asunto: Reducción de capacidad tras aceptación de la solicitud <<id_solicitud>> de tipo <<tipo solicitud>>, de <<actor_inicial>> • Cuerpo: Informarle que tras la aceptación de la solicitud <<id_solicitud>> de tipo <<tipo solicitud>>, de <<actor_inicial>>, se ha reducido a 0 la capacidad del/de los contratos

Código	RF06
Modulo	Notificaciones
Nombre	Modificación voluntaria de solicitudes
Descripción	Caso de apertura de segunda ventana para la modificación voluntaria de las solicitudes • Destinatarios: GTS, comercializador • Asunto: Apertura de Ventana para Modificación voluntaria de la solicitud <<id_solicitud>> de tipo <<tipo solicitud>>, de <<actor_inicial>> • Cuerpo: Informarle que debido a que no ha sido posible la asignación de slots por presentar incompatibilidades con otras solicitudes, se abrirá una Ventana para la modificación de la solicitud <<id_solicitud>> de tipo <<tipo solicitud>>, de <<actor_inicial: • fecha: <<fecha>> • hora inicio:<<hora inicio>>

- Para el módulo de Garantías los requisitos funcionales son los siguientes:

Código	RF01
Modulo	Garantías
Nombre	Evaluar los contratos sin garantías
Descripción	Operación que consultará en base de datos los meses de los contratos sin garantías, filtrando por las fechas indicadas para los casos de no Slot. Con los meses obtenidos se evalúa si esos contratos deben seguir teniendo garantía.

Código	RF02
Modulo	Garantías
Nombre	Evaluar los contratos sin garantías slot
Descripción	Operación que consultará en base de datos los meses de los contratos sin garantías filtrando por las fechas indicadas para los casos de Slot. Con los meses obtenidos se evalúa si esos contratos deben seguir teniendo garantía.

Código	RF03
Modulo	Garantías
Nombre	Obtener los meses de las garantías
Descripción	Operación que consultará en base de datos los parámetros administrados de meses para Slots y no Slot filtrando por los servicios según informen en la entrada. Para el caso de Slot: • CB(4, "CB", "Carga de Buques") • TBB(5, "TBB", "Trasvase de Buque a Buque") • PF(6, "PF", "Puestas en frío") • DB(29, "DB", "Descarga de Buques")

Código	RF04
Modulo	Garantías
Nombre	Eliminar las garantías pendientes
Descripción	Se eliminarán todas las garantías (dejarán de existir) cuyo último requerimiento este en estado pendiente.

3.3. Requisitos no funcionales

Una vez descritos los requisitos funcionales se pasa a hablar de los requisitos no funcionales ya que ambos influyen en las posteriores fases de desarrollo.

Los requisitos no funcionales, a diferencia de los funcionales, definen características que no tienen que ver directamente con el uso que van a hacer los usuarios del sistema, sino que son características que debe cumplir el sistema en conjunto para permitir a los usuarios aprovecharse de las funcionalidades que se les ofrece.

RNF01	Todo el código nuevo deberá pasar los requerimientos de calidad impuestos por el cliente y comprobados a través de Sonar.
RNF02	En el código no debe existir ninguna vulnerabilidad de seguridad, comprobado mediante Kiuwan.
RNF03	La aplicación deberá funcionar en Internet Explorer, en Firefox y en Google Chrome.
RNF04	Se deberá proporcionar una plataforma para el registro de incidentes.

4. Diseño

Después de haber detallado los diferentes requisitos, tanto funcionales como no funcionales definidos para los módulos a implementar, se procede a detallar el diseño para su construcción.

4.1. Diseño arquitectónico

El diseño de la arquitectura se basa en modelar el funcionamiento interno del sistema. El sistema a desarrollar es una aplicación web a la que se accede vía http mediante un usuario y una contraseña.

Dentro de la arquitectura podemos diferenciar entre varias capas:

- **Capa de datos**: donde como su propio nombre indica se alojan todos los datos de la aplicación, desde los datos de los clientes hasta la documentación de la aplicación.

- **Lógica de negocio**: construida principalmente sobre java utilizando spring como *framework*. Esta capa es el cerebro de la aplicación, o dicho de otro modo, la encargada de realizar toda la funcionalidad.

- **Presentación**: también conocida como la capa *Front-End*, es decir, la capa visual. En esta capa es donde se aprecia visualmente la funcionalidad realizada en la capa anterior. También es la única capa con la que interactúa y tiene contacto el usuario final.

4.2. Diseño detallado

Una vez especificada la arquitectura del sistema, se pasa a detallar el diagrama de lógica de negocio, pudiéndose visualizar en la *Figura 1*.

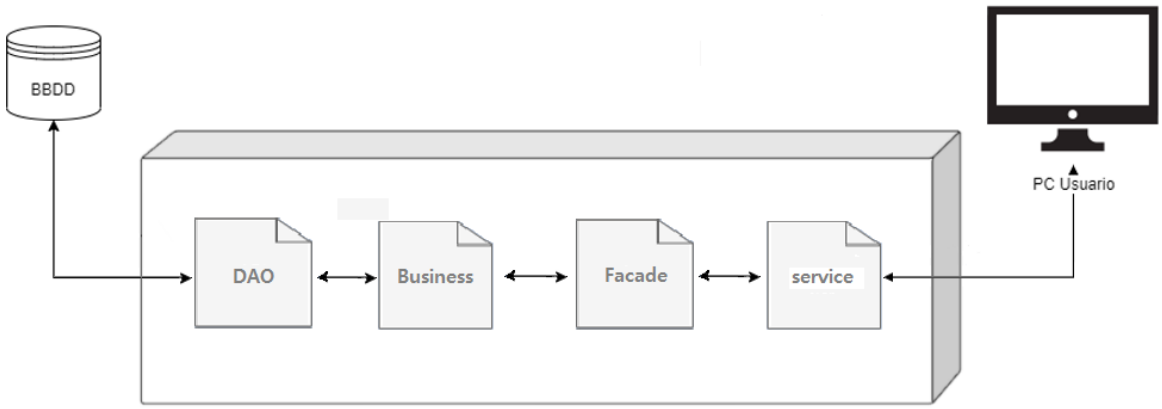


Figura 1: Diagrama de lógica de negocio

Como se puede apreciar en el diagrama, la aplicación contiene 3 módulos principales.

- El primer módulo corresponde con el servidor de base de datos, en el cual se almacena toda la información de la aplicación.

- El último es el PC del usuario, a través del cual se puede conectar a la aplicación, utilizando un navegador, para realizar todas las funcionalidades nuevas y las que ya estaban implementadas.

En estos dos grupos no vamos a entrar con más detalle, del que sí hablaremos es del de en medio, del desarrollo en sí de la lógica de negocio. Esto se va a detallar en el siguiente punto.

5. Implementación

Tras haber completado la captura de requisitos y explicado el diseño de la aplicación procederé a detallar la implicación del software. De las 3 capa habladas anteriormente yo me encargué del desarrollo de la lógica de negocio.

Para que se entienda correctamente seguiré el orden del flujo, empezando por la capa web y por petición expresa del cliente mostraré oculta información sensible del código.

5.1. Interfaz Service

Esta es la última capa de la aplicación Spring, la que interactúa con la capa front, realizando la conexión y el traspaso de datos a través de una url predefinida.

Esta capa es la encargada de procesar las peticiones realizadas desde el aplicativo web, es la que interactúa con la capa front y realiza la conexión y el traspaso de datos a través de una url predefinida. Como podemos observar en la siguiente imagen la interfaz contiene las anotaciones *@WebService* que nos indica que implementa un servicio web y *@WebMethod* indicando que ese método es una operación de servicio web con la propiedad *action* que define la acción de la operación.

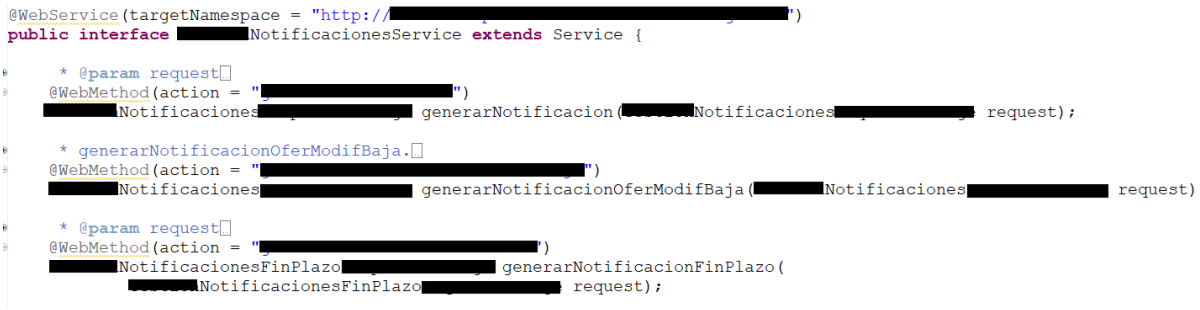
En mi caso se ha desarrollado únicamente un *service* por cada módulo realizado, es decir, dos *services*.

```
@WebService(targetNamespace = "http://service.[redacted].es")
public interface [redacted]GarantiaService extends Service {

    /**
     * @param request Mensaje que encapsula la petición efectuada al servicio.
     * @return [redacted]Garantia[redacted] Mensaje que encapsula la respuesta generada desde el servicio.
     */
    @WebMethod(action = "[redacted]")
    [redacted]Garantia[redacted] calcularGarantia([redacted]Garantia[redacted] request);

    /**
     * @param request Mensaje que encapsula la petición efectuada al servicio.
     * @return [redacted]Garantia[redacted] Mensaje que encapsula la respuesta generada desde el servicio.
     */
    @WebMethod(action = "[redacted]")
    [redacted]Garantia[redacted] liberacionGarantia([redacted]Garantia[redacted] request);
```

Figura 2: Interfaz Garantía Service



```
@WebService(targetNamespace = "http://[redacted]")
public interface [redacted]NotificacionesService extends Service {

    * @param request
    @WebMethod(action = "[redacted]")
    [redacted]Notificaciones[redacted] generarNotificacion([redacted]Notificaciones[redacted] request);

    * generarNotificacionOferModifBaja.
    @WebMethod(action = "[redacted]")
    [redacted]Notificaciones[redacted] generarNotificacionOferModifBaja([redacted]Notificaciones[redacted] request)

    * @param request
    @WebMethod(action = "[redacted]")
    [redacted]NotificacionesFinPlazo[redacted] generarNotificacionFinPlazo(
            [redacted]NotificacionesFinPlazo[redacted] request);
```

Figura 3: Interfaz Notificación Service

5.2. Implementación Service

Las interfaces tienen una serie de métodos que son desarrollados en las clases de implementación. Siguiendo la misma estructura, aportan la lógica a la capa. De esta manera, como podemos ver en la *Figura 3*, volvemos a tener las anotaciones *@WebService* y *@WebMethod*.

```
@WebService(targetNamespace = "http://service.[redacted].es")
public class [redacted]GarantiaServiceImpl implements [redacted]GarantiaService {

    private [redacted]GarantiaFacade [redacted]GarantiaFacade;
    private final Logger logger = LoggerFactory.getLogger([redacted]GarantiaServiceImpl.class);

    /**
     * constructor
     * @param [redacted]GarantiaFacade
     */
    public [redacted]GarantiaServiceImpl([redacted]GarantiaFacade [redacted]GarantiaFacade) {
        this.[redacted]GarantiaFacade = [redacted]GarantiaFacade;
    }

    /**
     * @param [redacted]Garantia[redacted]
     * @return [redacted]Garantia[redacted]
     */

    @WebMethod(action = "[redacted]")
    public [redacted]Garantia[redacted] calcularGarantia([redacted]Garantia[redacted] request) {

        return this.[redacted]GarantiaFacade.calcularGarantia(request);

    }

    /**
     * @param [redacted]Garantia[redacted]
     * @return [redacted]Garantia[redacted]
     */
    @WebMethod(action = "[redacted]")
    public [redacted]Garantia[redacted] liberacionGarantia([redacted]Garantia[redacted] request) {

        return this.[redacted]GarantiaFacade.liberacionGarantia(request);
    }
```

Figura 4: Implementación Garantía Service

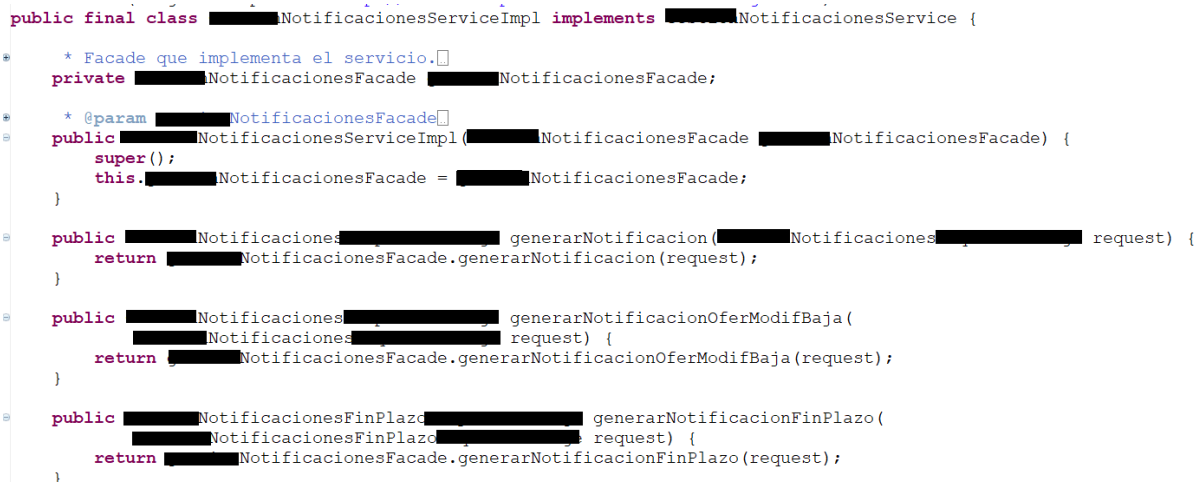


Figura 5: Implementación Notificación Service

5.3. Interfaz Facade

Desde la capa service se llama a la capa facade que vuelve a tener los mismos metodos con *@WebMethod.* Al tener los mismos métodos la llamada se hace uno a uno, es decir, *calcularGaratia* de *Service* llama a *calcularGarantia* de *Facade.*

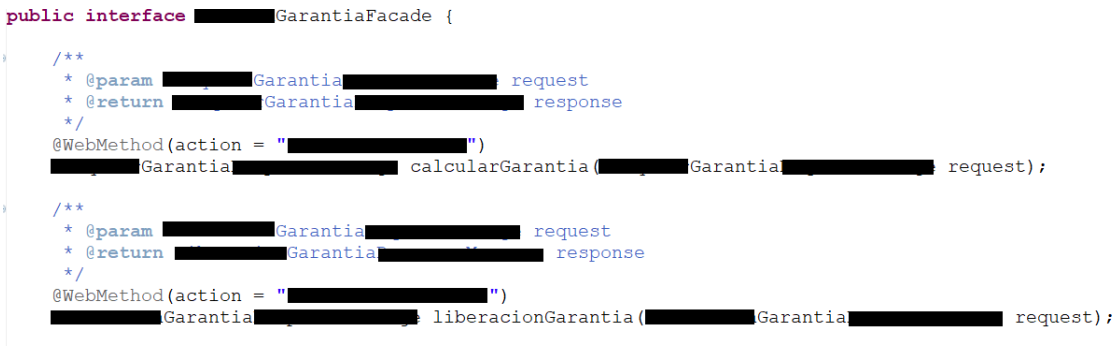


Figura 6: interfaz Garantía Facade

5.4 Implementación Facade

Como en el caso anterior, la implementación realiza la lógica de los métodos. En este caso con la anotación *@Transactional.*

```
public class [redacted]GarantiaFacadeImpl implements [redacted]GarantiaFacade {

    private [redacted]GarantiaBusiness [redacted]GarantiaBusiness;
    private [redacted]GarantiaPantallaBusiness [redacted]GarantiaPantallaBusiness;
    private [redacted]GarantiaDispBusiness [redacted]GarantiaDispBusiness;

    private final Logger logger = LoggerFactory.getLogger([redacted]GarantiaFacadeImpl.class);

    @Transactional(propagation = [redacted])
    public LiberacionGarantia[redacted] liberacionGarantia(LiberacionGarantia[redacted] request) {
        LiberacionGarantia[redacted] response = new LiberacionGarantia[redacted]();
        response = [redacted]GarantiaBusiness.liberacionGarantia(request);
        return response;

    }

    @Transactional(propagation = [redacted])
    public [redacted]Garantia[redacted] calcularGarantia([redacted]Garantia[redacted] request) {
        [redacted]Garantia[redacted] response = new [redacted]Garantia[redacted]();

        // control de datos no nulos
        this.controlObligatorioRequest(request);
        // [redacted]

        List<ContratoDTO[redacted]> contratosRechazados = new ArrayList<ContratoDTO[redacted]>();

        response.setListaGarantias([redacted]GarantiaBusiness.transformacionDeCampos(request.getContrato(), request
                .getOrigen(), request.getOperacion(), contratosRechazados));
```

Figura 7: Implementación Garantía Facade

```
/**
 * Método que genera una notificación dado un id de solicitud y un id de notificación
 */
@Transactional(propagation = [redacted])
public [redacted]Notificaciones[redacted] generarNotificacion([redacted]Notificaciones[redacted] request) {

    // Notificaciones [redacted]
```

```
@Transactional(propagation = [redacted])
public [redacted]Notificaciones[redacted] generarNotificacionOferModifBaja(
        [redacted]Notificaciones[redacted] request) {

    [redacted]Notificaciones[redacted] response = new [redacted]Notificaciones[redacted]();
    response.setOk(true);
```

```
/**
 * Método que genera una notificación de fin de plazo dado una lista de id de solicitud y un id de notificación
 */
@Transactional(propagation = [redacted], timeout = 1200)
public [redacted]NotificacionesFinPlazo[redacted] generarNotificacionFinPlazo(
        [redacted]NotificacionesFinPlazo[redacted] request) {
```

Figura 8: Implementación Notificación Facade

5.5 Interfaz Business

De las capa facade se llama a la business como se observa en la *Figura 6*.

```
public interface [redacted]GarantiaBusiness {

    /**
     * @param request LiberacionGarantia[redacted]
     * @return LiberacionGarantia[redacted]
     */
    LiberacionGarantia[redacted] liberacionGarantia(LiberacionGarantia[redacted] request);

    /**
     * Calcula valorGarantia a partir del DTO (con 2 dígitos decimales).
     * @param garantia[redacted] los campos [redacted]
     */
    List<[redacted]GarantiaDTO> transformacionDeCampos(List<Contrato[redacted]> listaContrato[redacted], Origen[redacted] origen,
            Operacion[redacted] operacion, List<Contrato[redacted]> contratosRechazados);
```

Figura 9: Interfaz Garantía Business

5.6 Implementación Business

Del mismo modo, nos encontramos con interfaz e implementación, siendo la implementación el desarrollo de la clase.

```
public class [redacted]GarantiaBusinessImpl implements [redacted]GarantiaBusiness {

    private [redacted]GarantiaDAO [redacted]GarantiaDAO;
    private [redacted]DAO [redacted]DAO;
    private [redacted]DAO [redacted]DAO;
    private [redacted]DAO [redacted]DAO;
    private final Logger logger = LoggerFactory.getLogger([redacted]GarantiaBusinessImpl.class);

    /**
     * @param request LiberacionGarantia[redacted]
     * @return LiberacionGarantia[redacted]
     */
    public LiberacionGarantia[redacted] liberacionGarantia(LiberacionGarantia[redacted] request) {


    public List[redacted]GarantiaDTO> transformacionDeCampos(List<Contrato[redacted]> listaContrato[redacted], Origen[redacted] origen,
            Operacion[redacted] operacionAux, List<Contrato[redacted]> contratosRechazadosAux) {
```

Figura 10: Implementación Garantía Business

5.7 Interfaz DAO

Por último, se llama a la capa DAO

```
public interface [redacted]GarantiaDAO extends DAO {

    /**
     * Obtiene los códigos de parámetros ([redacted]_PARAMETRO.COD_PARAMETRO) y los valores de garantía
     * ([redacted]_GARANTIA.TXT_VALOR) [redacted]
     * [redacted].
     * @param servicio
     * @param buque [redacted].
     * @param peaje [redacted]
     * @param fecInicioContrato
     * @param interrumpible
     * @return
     */
    List<[redacted]GarantiaDTO> recuperarParametros([redacted] servicio, [redacted] buque, [redacted] peaje,
            Date fecInicioContrato, [redacted] interrumpible);
```

Figura 11: Interfaz Garantía DAO

5.8. Implementación DAO

Esta última capa es la encargada de realizar las transacciones contra la base de datos de la aplicación.

Además tenemos Parsers, que sirven para pasar los datos que nos llegan de base de datos a DTO siguiendo el patrón de diseño DTO (Data Transfer Object), y usando para ello la herramienta Hibernate, siendo Hibernate una herramienta de mapeo objeto-relacional para Java que permite el mapeo de atributos entre los objetos de una aplicación y la base de datos. Con el uso de sus anotaciones se realizó el mapeo objeto-relacional, creando una clase entidad por cada tabla.

```
public class [redacted]GarantiaDAOImpl extends DAOImpl implements [redacted]GarantiaDAO {

    private final Logger logger = LoggerFactory.getLogger([redacted]GarantiaDAOImpl.class);
    private final String errorIntegridad = "Alguno datos no cumplen las restricciones de integridad referencial";
    private final Integer HORA = 23;
    private final Integer MINUTOS = 59;
    private final Integer SEGUNDOS = 59;

    private ParserGarantia[redacted] parserGarantia[redacted];
    private ParserGarantia[redacted] parserGarantia[redacted];
    private Parser[redacted] parser[redacted];
    private Parser[redacted] parser[redacted];

    public List<[redacted]GarantiaDTO> recuperarParametros([redacted] servicio, [redacted] buque,
            [redacted] peaje, Date fecInicioContrato, [redacted] interrumpible) {

        logger.debug("Entra en el método [redacted]GarantiaDAOImpl.recuperarParametros");
```

Figura 12: Implementación Garantía DAO

6. Pruebas

La última fase del proceso de ingeniería software que se describe en esta memoria es la de pruebas. Este capítulo recoge las diferentes pruebas que se han realizado con relación a los módulos desarrollados.

6.1. Unitarias

Las pruebas unitarias permiten verificar el correcto funcionamiento de cada uno de los módulos del sistema por separado. A través de estas pruebas se verifica que la lógica de negocio se encuentra correctamente implementada. Esto se llevo a cabo mediante varias librerías como por ejemplo *JUnit, Mockito* y *powerMockito*.

Con *Junit* se desarrollaron los test para comprobar la correcta funcionalidad de los diferentes métodos. Usando *Mockito* en gran cantidad de esos test ya que tuvimos que simular el funcionamiento de muchos métodos y llamadas a clases externas para que un fallo en esos métodos y llamadas no diese error y dejase de ser el test unitario. El *powerMockito* fue utilizado para testear métodos privados los cuales no se podrían mockear de otra forma.

Para pasar estas validaciones era necesario llegar a cubrir un 70% de cobertura en los test por parte del cliente, aunque nosotros queríamos superar el 80% para mayor seguridad.

6.2. Integración

En segundo lugar, se encuentran las pruebas de integración cuyo objetivo es comprobar que todos los componentes de la aplicación funcionen correctamente actuando en conjunto. Para eso hemos utilizado la herramienta SoapUI generando peticiones SOAP donde se insertan los parámetros deseados para la realización de las pruebas.

6.3. Calidad

Las pruebas de calidad fueron realizadas mediante *SonarQube*. Esta herramienta se usa para comprobar la cobertura del código cubierta por los test unitarios mencionados anteriormente.

Además de eso, se comprueba la cantidad de comentarios que se encuentran en el código, las vulnerabilidades presentes, la correcta nomenclatura de atributos, métodos y clases y la cantidad de código repetido, entre otras cosas.

Los parámetros de estas comprobaciones vienen dados por el cliente, pues es él quien marca los mínimos necesarios de cada comprobación dentro de *SonarQube*.

6.4. Aceptación

Este tipo de pruebas consisten en verificar que el sistema cumple con los requisitos y objetivos definidos al inicio del proyecto, y que por tanto, se satisfacen las necesidades del cliente.

Para la realización de estas pruebas se debe haber desplegado el proyecto en desarrollo. Además se cuenta con un documento con todas las pruebas que se deben realizar paso por paso. Dichas pruebas son realizadas, en primer lugar, por el equipo de desarrolladores del que formo parte. Una vez superadas todas las pruebas del documento, el proyecto era desplegado en integración y se pasa a probar por el propio equipo de pruebas del cliente, quien debería volver a realizar todas las pruebas del documento, comprobando su correcto funcionamiento. Si alguna de estas pruebas resulta errónea, se cataloga el error como una incidencia y debemos solucionar el problema sin ningún cargo económico.

En esta fase de pruebas obtuvimos dos incidencias y dos cambios a nivel de requisitos. Por mi parte yo me ocupé de las incidencias y mi compañero de los cambios. Las incidencias fueron corregidas y los cambios fueron desarrollados sin coste alguno para la otra empresa ya que fueron cambios en validaciones muy simples. Tras haber corregido las incidencias y los cambios, el cliente dio su aprobación tras el éxito de estas pruebas.

6.5. Despliegue

De esta última parte yo realicé un trabajo pasivo, pues era mi compañero el que se encargaba de esto, pero realizó todo el proceso conmigo explicándome cada paso.

Los despliegues se realizaban en servidores WebLogic de Oracle, pero solo tras haber obtenido el permiso del cliente. Estos despliegues podían hacerse tanto en el entorno de desarrollo o como en el de producción. Dependiendo del despliegue que se

quisiese hacer, se accedía a un servidor o a otro. Primero debíamos meter los archivos generados tras la compilación del componente API en las carpetas del servidor. Para más tarde, acceder a la consola del WebLogic y desplegarlos en el servidor.

El servidor de producción era el servidor principal que alojaba la actual aplicación en uso para los usuarios del cliente, por lo que este servidor no podía tener ningún tipo de fallo.

De este modo, el servidor de desarrollo, era el dedicado a pruebas, pues aquí se comprobaba que no se hubiera dejado ningún fallo en todas las pruebas anteriores, de ser así, el fallo era corregido, se volvían a pasar todas las pruebas anteriores y por último se repetía el despliegue en desarrollo. Solo cuando se ha comprobado que todo funciona correctamente en desarrollo, es cuando se realiza el despliegue en producción.

7. Conclusiones y trabajos futuros

En el último capítulo se recogen las conclusiones a las que se ha llegado tras el desarrollo de este proyecto y se enumeran una serie de trabajos futuros que han ido surgiendo como ideas y que aún se desconoce si serán implementados en futuras versiones de la plataforma.

7.1. Conclusiones

El objetivo del proyecto era desarrollar el módulo de notificaciones y el de garantías, cumpliendo con todos los requerimientos de la empresa gasística, tanto de desarrollo, como del tiempo de entrega. Los módulos fueron finalizados y desplegados con éxito dentro del plazo establecido tras superar las diferentes pruebas y comprobar que el funcionamiento era el esperado.

A nivel personal, este desarrollo me ha servido para afianzar conocimientos aprendidos en la carrera, como es el caso de java y git. Además de profundizar más en toda la parte de software debido a que yo no cursé esta mención. También cabe añadir el aprendizaje de tecnologías nuevas como es el caso de sprint o incluso algo de angular.

Por otro parte, el haber trabajo en un proyecto real me ha servido para coger experiencia en la vida laboral, de cara a trabajar en una empresa, tener que hablar con el cliente, o saber trabajar en paralelo con otros compañeros.

Tras la realización de las prácticas y gracias al trabajo que he realizado durante todo el proyecto he pasado a formar parte de la empresa como trabajadora.

7.2. Trabajos futuros

Debido a que el trabajo realizado eran dos módulos de un proyecto de mayor envergadura, los trabajos futuros tienen que ver con el desarrollo de nuevos módulos, por ejemplo, actualmente se está hablando de empezar a desarrollar el módulo de gestión de comunicación.

8. Bibliografía

Todas las referencias son online y los enlaces se consultaron por última vez el 9 de septiembre de 2020

[1] Java

https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[2] Spring

https://es.wikipedia.org/wiki/Spring_Framework

https://spring.io/projects/spring-framework

[3] Maven

https://maven.apache.org/

https://es.wikipedia.org/wiki/Maven

[4] Soap UI

https://www.soapui.org/soapui-projects/soapui-projects.html/

https://es.wikipedia.org/wiki/SoapUI

[5] Svn

https://lihuen.linti.unlp.edu.ar/index.php/C%C3%B3mo_usar_SVN

[6] MobaXterm

https://mobaxterm.mobatek.net/

[7] Oracle WebLogic

https://www.oracle.com/es/middleware/weblogic/

[8] Oracle Sql developer

https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html

[9] Jira

https://www.atlassian.com/es/software/jira

[10] Microsoft teams

https://es.wikipedia.org/wiki/Microsoft_Teams

https://www.microsoft.com/es-es/microsoft-365/microsoft-teams/group-chat-software